SOFTWARE ANALYSIS SYSTEM HAVING AN APPARATUS FOR SELECTIVELY COLLECTING ANALYSIS DATA FROM A TARGET SYSTEM EXECUTING SOFTWARE INSTRUMENTED WITH TAG STATEMENTS AND METHOD FOR USE THEREOF

TECHNICAL FIELD

The present invention relates to software testing and debugging, and more particularly, to collection of analysis information from target systems executing software instrumented with executable tag statements.

BACKGROUND OF THE INVENTION

Creating executable software code typically involves a software developer first creating a source code program with a text processing program, followed by compiling and linking the source code to create executable code for a specified computer processor. The executable code is subsequently stored in an executable file. The executable code can then be debugged by the software developer by executing the executable file on the specified computer processor to determine if the software performs its tasks correctly, or if instead one or more errors occur during execution. If the executable code has errors, the software developer can modify the source code in an attempt to remove the errors, recompile the source code, and then link the recompiled code to produce a new executable file for debugging. For large software programs, this process is repeated several times until all known errors are removed.

The debugging process, which is quite often a very involved process, many times requires the software developer to locate the cause of an error from executing the executable file. Various methods exist for a software developer to identify errors. In one such method, a software developer adds print statements throughout the source code so that as the executable file is executing, the corresponding executable print instructions are also executed to report the current progress of the execution. Knowledge of the current execution progress assists the software developer to identify the section of the code that is executing when an error occurs. Additionally, print statements can be used to also display the current value of variables or source code expressions at specified points throughout the execution. Since the print statements are part of the original compilation and linking process, the variables and expressions that are part of the print statements are evaluated in the context of the current variable scope, as would any other compiled code statement. An example of this is using the value of a local variable in a currently executing function rather than a variable with the same name in a different non-executing function.

In addition to print requests, application programs known as debuggers may be also used by the software developer to assist with locating errors in the executable code. A debugger loads executable code into memory and provides additional control over execution of executable files. For example, the software developer can use the debugger to execute one executable code instruction at a time. Alternately, the debugger may be used to execute the executable code continuously until a break point, designated by the software developer within the debugger, is reached. When execution of the executable code is stopped, a user can interact with the debugger to view current values of variables and expressions. Some debuggers can also reconstruct the source code from information stored in an executable code file during the compiling and linking steps, and display the source code lines that correspond to the instructions in the executable code. The display of the source code facilitates control by the software developer of the execution of the executable code.

Software developers debugging source code written for embedded systems face particular challenges when trying to analyze the performance of the software. An embedded system may be characterized as one whose primary purpose is to perform a specific function rather than to perform general computational functions. A microprocessor-based microwave oven controller, a microprocessor-based automobile ignition system, and a microprocessor-based telephone switching system are all examples of embedded systems. The techniques of debugging software discussed above are not easily applied to software written for embedded systems. One reason is that many embedded systems include only a central processor and limited memory, and do not include access to other standard computer system input or output devices, such as a keyboard or display. As such, it is often difficult for a software developer to debug or analyze the performance of software written for these types of systems using the conventional techniques. However, fortunately for the software developer, there are several alternative techniques, and software analysis equipment, that are more suited for analyzing software written for and executed in these types of systems.

One such technique uses executable code marker statements inserting into spaces created in the source code during compilation. That is, the source code is prepared for debugging by compiling the source code and inserting empty spaces following each function of the source code. The empty spaces are reserved for inserting the code markers during an analysis phase by the software developer after the source code has been compiled and linked. The inserted code markers are captured by a debugging system during the execution of the modified compiled source code, and the results from the collection of code markers are used as a means of determining the performance of the software program. For example, each tag statement may write a value to a respective address so that the identity of the address containing that value provides an indication of which tag statements were executed. The data is collected by the host system and displayed in a manner that assists the software developer with debugging and evaluating the performance of the software. A more detailed description of this technique and analysis system is provided by U.S. Patent No. 5,265,254, to Blasciak *et al.*

Another technique and analysis system employs a host system coupled to the microprocessor of an embedded system. Executable tag statements inserted into a source code are executed along with the compiled and linked source code. Each of the tag statements cause the microprocessor to write a tag to a predetermined address location in a memory of the embedded system. A probe coupled to the external terminals of the embedded system monitors an address bus for the predetermined address, and when detected, latches data provided by an executed tag statement from a data bus. Based on the value of the data latched, the analysis system is able to determine the location in the source code being executed. A more detailed description of this technique and analysis system is provided by U.S. Patent No. 5,748,878, to Rees *et al.*

Although the aforementioned techniques and analysis systems provide the software developer with powerful tools for debugging and evaluating the software for embedded systems, it is difficult in these systems to modify the type of data collected during the execution of the instrumented source code. These systems typically allow the software developer to manually insert a limited number of executable statements into the executable code after compilation and linking. However, where the modifications to the executable statements are extensive, or the source code is long, such a technique is inefficient and impractical. Thus, adding or deleting executable statements from the executable code often requires the software developer to instrument, compile, and link the source code each time changes are made.

SUMMARY OF THE INVENTION

The present invention relates to a software analysis system for analyzing software executing on a target system. The software analysis system includes a filter to selectively collect tags emitted by the target system during execution of the software instrumented with executable tag statements. The tags are captured and the decoded tag type of the captured tag is compared to a programmable filtering criteria to determine whether tag data of the captured tag should be collected and processed. The programmable filtering criteria may be implemented by a filter table that stores a collection flag for each tag type. The collection flag is indicative of whether the respective tag types should be collected when emitted from the target system. The programmable filtering criteria may be modified by collecting filter tags emitted by the target system during execution of the instrumented software, or by direct modification by a software developer through a user interface.

According to another aspect of the present invention, the analysis system includes a tag buffer to store the collected tag data prior to processing.

According to another aspect of the present invention, the analysis system includes a timestamp generator for appending timestamp information to the tag data when stored in the tag buffer.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is an isometric view of a software analysis system according to an embodiment of the present invention.

Figure 2 is a schematic and block diagram of the software analysis system of Figure 1 and its manner of use.

Figure 3 is a schematic and block diagram of a trace buffer according to an embodiment of the present invention.

Figure 4 is a flowchart of an operation of the trace buffer of Figure 3.

DETAILED DESCRIPTION OF THE INVENTION

One embodiment of a software analysis system 10 in accordance with the present invention is illustrated in Figure 1. The system 10 includes a probe tip 12 that clips onto the microprocessor of a target system (not shown) in a conventional manner. As a result, the external connector pins of the target system microprocessor, including its data bus and address bus, are accessible to the probe tip 12. The probe tip is connected through a conventional ribbon conductor 18 to a probe chassis 20 containing most of the electronics for the system 10. The probe chassis 20 is, in turn, connected through a suitable cable 30, such as an Ethernet cable, to a host system 40. The host system 40 is essentially a conventional PC computer having a processor chassis 42 with a disk drive 44, a CRT monitor 46 with a display screen 48, and a keyboard 50. The host system 40 preferably uses a Unix® or Windows® user interface and operating system. Application specific software is loaded through the disk drive 44 to cause the host system 40 to properly interface with the probe chassis 20, receive appropriate configuration and operating commands through the keyboard 50, and display analysis results on the screen 48.

The use of the software analysis system 10 is illustrated in Figure 2. Source code 60 written to run on a target system is first instrumented by inserting tag statements 62 in the source code 10 at various locations that the user is interested in analyzing. For example, if the user is interested in determining code coverage, the user will insert a tag statement 62 in each branch of the source code 60, and the system 10 will determine which of the branches have been executed based on whether each tag statement has been executed. Other types of analysis functions are described in detail in the Rees *et al.* patent, which is incorporated herein by reference. The insertion of tag statement 62 in the source code 60 results in instrumented source code 64. When the instrumented code 64 is produced, a symbol database 65 is also created, which provides a record correlating each of the tag statements to their locations in the source code 10. The instrumented source code 64 is compiled in a conventional manner at 66 thereby resulting in executable code 68. The executable code 68 is then loaded into the target system T by any suitable means. For example, the executable code may be stored in a programmable read-only memory ("PROM") that is installed in the target system T. The executable code 68 may also be executed in the target system T through a conventional emulator (not shown). Regardless of how the executable code 68 is loaded into the target T, the target T is then allowed to execute the code. The probe tip 12 clips on to the target system T in a conventional manner to make electrical contact with at least the address bus and the data bus of the target system T. Tags generated by the execution of tag statements 62 and collected by the probe tip are transferred to the probe chassis 20 through ribbon cable 18. After the probe chassis 20 has filtered and processed the data from the probe tip 12, it outputs appropriate data to the host system 40 through the local area network cable 30.

Host application software 70 includes processing routines 72 that store data in and retrieve data from data files 74, and the host application software 70 also includes a graphical user interface 75, such as the X-11 or Microsoft Windows® interface, that works with the processing routines 72 to operate on the data files 74 and provide various displays of analysis data. The processing routines 72 also receive the symbol database 65 so that the tag execution data in the data files 74 can be correlated with the location of the tag statements in the source code 65 in order to provide reports and displays that specify performance in terms of source code locations and branches. The symbol database 65 is preferably loaded into the host through the disk drive 44 (Figure 1). The host application software 70 also includes data structure 76 for storing and handling the analysis data, and communications software 78 for providing communication with the target access probe 20.

In operation, each of the tag statements 62 generate a respective tag containing a data field having a "tag value" that is generally unique to the location of the tag statement in the source code 60. Thus, for example, a first branch may contain a tag statement having a tag value of 1. A second branch may contain a tag statement having a tag value of 2, and so forth. When the tag statement 62 is executed by the target T, a processor in the target T writes a tag containing the tag value to a predetermined location in the address space of the target system T, also known as a tag port address. As explained in greater detail below, the tag 62 may also contain at least one other field providing information about its function or location of its associated tag statement 62 in the source code 60. More specifically, the tag statement 62 preferably writes a tag consisting of 32 bits which includes not only a data field word having a tag value, but also a number of bits which define the type or category of tag. For example, different tag types may identify function entry and exit points, branch points, and memory allocation statements. Tags having a tag type field to identify the tag type are known as "control tags." In the preferred embodiment of the system 10, all control tags are written to the same tag port address.

The system 10 also utilizes data tags. Data tags accompany control tags and are written to a second tag port address to provide additional information relevant to a particular control tag. For example, a control tag may indicate that a memory allocation is taking place, and two data tags accompanying the control tag may indicate the size of the memory allocation and the memory pointer associated with that allocation, respectively. Since only a single location in the address space of the target system preferably is used for control tags and a relatively few locations used for data tags, the preferred embodiment of the inventive system 10 does not significantly use the memory resources of the target system, thus making the analysis system substantially transparent to the target system.

In addition to the control and data tags described in U.S. Patent No. 5,748,878 to Rees *et al.*, the embodiments of the present invention further include a filter tag, which is used to modify the criteria for collecting the tags from the target system T. The filter tag is itself a special type of control tag. The system 10 can collect (i.e., write to a tag buffer) specific types of tags from the target system, rather than collect every tag from the target T during the execution of the instrumented software program, as with the software analysis system described in detail in the aforementioned patent to Rees *et al.* As will be explained in further detail below, all of the possible types of control and data tags, as well as an associated collection flag indicating whether the particular tag type should be collected, are stored in a filter table. The filter tags instrumented into the source code, and emitted by the target system during execution, may be used by the system 10 to modify the status of the collection flags for the various tag types. Thus, collection of specific tag types may be altered while the software program is being executed by the target system.

Including a collection filtering mechanism in the system 10 that can be used to reduce the overall data collected by the trace system provides many benefits. One such benefit is that the required storage resources may be reduced because only the tag types desired by the software developer are collected by the system 10.. For a hardware based collection mechanism, that is, where tags are collected from the target system T through a probe and stored in a dedicated hardware buffer, there are lower material costs resulting from the decrease in the required buffer size. For a software collection mechanism that utilizes the memory of the target system as a tag buffer, less target system memory is used, and consequently, consumption of target resources is reduced. Generally, the hardware and software resources of the system 10 are better utilized. Another benefit is provided by way of faster post-processing time of the tag data since the amount of data collected by the system 10 is reduced to only the desired tag types. As a result, the display time of relevant information may be reduced, and the relevant information can be more clearly displayed. It can be appreciated that additional benefits may be provided by the embodiments of the present invention.

A tag data filtering method was briefly discussed in the Rees *et al.* patent. However, as described therein, the filtering method was related to only filtering the display of analysis data. In the trace system of the Rees *et al.* patent, the tag data is unconditionally collected by the software analysis system and the software developer can then filter the display of the unconditionally collected tags such that only the functions of interests are displayed. In contrast, the embodiments of the present invention can be programmed to selectively collect particular tags from the ones emitted by the Target system T during the execution of the instrumented software program.

The probe tip 12 monitors the address bus and the data bus of the target T and determines when the processor addresses the tag port addresses of the address space of the target system T. The probe tip 12 then captures the tag value currently on the data bus. As a result, the currently captured tag value indicates the location in source code 60 currently being executed. Moreover, the system 10 monitors the execution of the software in the target T in essentially real time since the probe 20 receives each of the tag values as it is captured.

With the information collected by the probe chassis 20, the system 10 is capable of determining function and task execution times, coverage analysis (i.e., identifying portions of the source code executed or not executed), memory allocation analysis (i.e., identifying how much memory each allocation statement in the source code allocates and identifying specific allocation errors), and program tracing (i.e., creating a sequential history of the execution of the source code). Finally, the probe chassis 20 communicates with the host system 40 to upload the data and allow it to be displayed by the host system 40.

The software analysis system 10 of Figures 1 and 2 is shown in greater detail in the block diagram of Figure 3. The probe chassis 20 includes a communications and control circuit 132 coupled to the host system 40 through the local area network cable 30 (Figure 2). As mentioned previously, the interface between the probe chassis 20 and the host system 40 may consist of a standard Ethernet communication channel. The probe chassis 20 is further coupled to the target system T through the probe tip 12. As also mentioned previously, the probe tip 12 clips onto the target system T. Consequently, the probe tip 12 is usually specific to the particular microprocessor used by the target system T.

The software analysis system 10 has so far been described as being similar to the one described in the Rees *et al.* patent. However, the embodiments of the present invention have the capability to selectively collect specific tag types from the tags unconditionally emitted from the target system T during execution of the software program. Shown in Figure 3 is an embodiment of a trace buffer 130 according to the present invention which is used to implement the tag filtering feature. The trace buffer 130 includes a tag register 134 that is programmed by the communications and control circuit 132 with the tag port addresses corresponding to the predetermined locations in the address space of the target system T. Typically, one tag port address is programmed for control tags and another tag port address is programmed for data tags. A tag filter 140 is coupled to the communication and control circuit 132 through multiplexers 142 and 144 for initialization and programming while the software analysis system 10 is disabled from the target system T.

A filter table 141 is implemented by the tag filter 140. The filter table 141 includes all of the possible types of control and data tags. Along with each tag type entry in the filter table 141 is a flag indicating the current collection criteria for that particular tag type. The filter table 141 is consulted by the tag filter 140 to determine which tag types emitted from the target system T will be collected. For example, if the flag associated with a particular tag type is set, that tag type will be collected. However, if the flag is not set, the tag type will be ignored although emitted from the target system T. Consequently, only those tag types having flags set as collectable in the filter table will be stored in a tag buffer 148. As will be discussed in greater detail below, the flags of the filter table 141 may be modified by either filter tags instrumented into the source code and collected by the trace buffer when emitted from the target system during the execution of the software program, or by the software developer directly accessing the filter table 141 through a filtering user interface.

When the software analysis system 10 is enabled, a TRACE ON signal switches the multiplexers 142 and 144 so that the tag filter 140 receives information from the data bus 150 and an output of a comparator 152, respectively. The comparator 152 generates a TAG HIT signal when an address detected on an address bus 154 matches one of the tag port addresses programmed in the tag address register 134. The TAG HIT signal is provided to the tag filter 140 and indicates that the tag presently on the data bus 150 should be preliminarily collected. A tag decoder 158 decodes the tag type of the tag data preliminarily collected and provides the tag type to the tag filter 140 to be compared with the current settings programmed in the filter table 141. If the tag type is determined as being one that should be collected by the system 10, the tag filter 140 provides a STORE signal to the tag buffer 148 indicating such. A timestamp generator 160 appends a timestamp to the tag data prior to being written into the tag buffer 148. The tag data and appended timestamp are eventually provided to the communications and control circuit 132 on the bus 136. The tag types determined as being not collectable are ignored by the system 10 and are not written to the tag buffer 148.

In contrast to receiving data tags, when filter tags are received by the system 10 on the data bus 150 and decoded by the tag decoder 158, they are written into the tag filter 140 instead of the tag buffer 148. The tag data of the filter tags are used to immediately update the status of the collection flags stored in the filter table 141 with new filtering information. Thus, the type of tag data collected and written to the tag buffer 148 will be modified as the software analysis system 10 is evaluating the target system T when a filter tag is collected.

A person of ordinary skill will appreciate that the chassis 20 also includes other conventional components for providing communication between the host 40 and the target system T. For example, a substitution memory into which the target system software is downloaded from the host 40 may be included, as well as communication ports that provide the host 40 and the target system T with a communication path upon which the two can communicate during the analysis. However, a detailed description of these components and their illustration in Figure 3 have been omitted in the interests of brevity.

By using filter tags, the software developer would generally have to instrument the source code only once, and then use the filtering mechanism to fine tune the desired trace criteria. The software developer could instrument the source code with any variety of tag types, such as program tracing tags, data tracing tags, and memory allocation tags. However, during the software analysis session the software developer can dynamically filter out undesired information by using filter tags to modify the filter table so that the system 10 collects only the desired tag types. For example, the software developer may instrument the source code for program tracing, data tracing, and memory allocation. However, if program tracing is the only information currently desired by the software developer, the filter table may be modified by having the target system emit the appropriate filter tag so that only the program tracing tags are collected by the system 10.

The filter tags may be placed in the target system in a number of different ways. For example, the filter tags may be instrumented into the source code using an instrumentor program. The software developer may also manually place filter tags into the source code. The filter tags may be nested in conditional statements so that the filter tags would be emitted by the target system T only if certain runtime conditions are satisfied. Alternatively, the filter tags may be placed into the target system during the analysis stage using the user interface of the host system 40.

Operation of the trace buffer 130 is described with respect to flowchart 200 shown in Figure 4. At a step 202, the tag filter 140 is initialized by the communications and control circuit 132, and the filter table 141 is configured to indicate which tag types should be initially collected by the software analysis system 10. The system 10 is enabled at step 204, and at step 206 the trace buffer 130 begins monitoring the address bus 154 for addresses matching any of the tag port addresses programmed into the tag address register 134. When a tag port address is detected at step 208, the tag data presently on the data bus 150 is decoded by the tag decoder 158 to determine its tag type at step 210. At step 212, the tag filter 140 compares the tag type with the settings of the filter table 141 to determine whether the tag should be collected and written into the tag buffer 148. If the tag type is not to be collected, the software analysis system 10 returns to step 204 to determine if the trace system remains enabled and then continues to monitor the address bus 154 for the next occurrence of a tag port address at step 206. However, for those tag types that should be collected, the tag filter 140 will then determine whether the tag presently on the data bus 150 is a filter tag at step 214. Filter tags are provided at step 216 to the tag filter 140 for immediately updating the filter table 141. The system 10 subsequently returns to step 204 to determine if the trace remains enabled, and at step 206 continues monitoring the address bus 154 if the determination at step 204 is true.

Tag types that should be collected, but which are not filter tags, have a timestamp appended to the tag data at step 218 by the timestamp generator 160. The timestamp information will be used to provide information related to the relative timing of the receipt of the tags, and consequently, the relative timing of execution by the target system T. At step 220, the tag and its appended timestamp are written into the tag buffer 148 and will be subsequently read by the communications and control circuit 132. The system 10 then returns to step 204 and will continue to monitor the address bus and collect specific tag types until the system 10 is disabled from the target system T or until execution of the software program is completed.

In an alternative embodiment of the present invention, the filtering criteria programmed in the filter table 141 may also be directly modified by the software developer using the system 10. That is, the software developer could directly access the filter table 141 through a user interface and set which tag types should be collected by the system 10 when emitted by the target system T. A user interface, similar to that described in detail in the Rees *et al.* patent, could be integrated as a menu item into a command window of the system 10. For example, a filtering user interface could display a table representing the filter table 141 which contains all of the possible types of control and data tags available for collection by the system 10 and the current status of the collection flags for each of the tags. The software developer could then select which tag types the system 10 should collect by selectively setting the collection flags accordingly. All tag types not having the collection flag set for collection are ignored by the system 10 when emitted by the target system T.

The features of the embodiments of the present invention may also be used in conjunction to provide the software developer greater flexibility in analyzing software. For example, by using the filtering user interface, the software developer can directly access the filter table 141 and set whether the system 10 should collect or ignore any filter tags coming from the target system T. Where the collection flag for the filter tags is set so that filter tags emitted by the target system T are to be ignored, the tag collection criteria will remain the same until the software developer directly modifies the filter table 141 at a later time. However, where the collection flag for the filter tags is set so that filter tags are to be collected, the collection criteria will be modified each time a filter tag is emitted by the target system T.

From the foregoing it will be appreciated that, although specific embodiments of the invention have been described herein for purposes of illustration, various modifications may be made without deviating from the spirit and scope of the invention. Accordingly, the invention is not limited except as by the appended claims.